Application of CGNS software components for helicopter blade fluid-structure strong coupling

M. Poinot, M. Costes, B. Cantaloube
ONERA
Châtillon, France

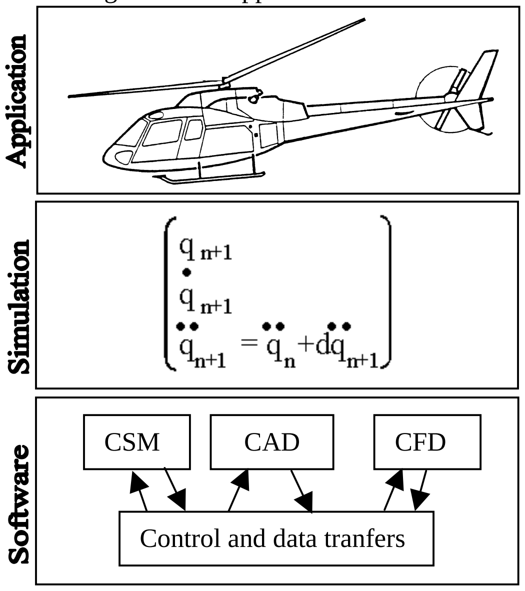
Abstract

The present paper describes the CGNS component approach we have developed and used for a fluid-structure code-coupling application. The CGNS standard provides a public data representation that can be used for data archival but also for data exchange. In the case of code-coupling, we show how a code can read and write CGNS compliant simulation data. Such a standard interface definition can help the interchange of software modules, for example the CSM module or the mesh deformation module can be interchanged with other modules.

Introduction

Among the various aeronautic configurations, the helicopter is certainly the one that involves the broadest interactions *between* its elements (main rotor, tail rotor, fuselage, wakes, jet exhaust, air intakes…). In order to be representative of the actual problems which the design engineer has to consider, the simulation has to capture these phenomena using a set of multidisciplinary solvers. Moreover, as the numerical techniques are improving and the computing platforms are getting more and more powerful, the scope and the complexity of the simulation grows. For efficiency, as well as in order to be consistent with the accumulated knowledge, the helicopter scientist wants to gain advantage of his past experience with simplified configurations close to the more complex one he is considering afterwards, and he also wants to reuse existing solvers, software translators, and any kind of software parts he had written. As a matter of fact, most simulations are done using already existing softwares, CFD solvers or structural dynamics solvers for example, but gluing them together is not straightforward.
Indeed, the building of a numerical simulation requires know-how both in the field of the helicopter aerodynamics and in the field of the software techniques. Although the scientist would like to spend most of his time working in the simulation level and trying to model the physical behaviour of the helicopter, more and more time is actually spent at the software level, because of the increasing complexity of the simulations. This is particularly the case when multidisciplinary applications are considered, requiring to build adapted interfaces between various computer codes which are to be run sequentially.
Two approaches can be found in order to reduce the related cost: (1) use a dedicated simulation workbench or (2) use standard components. The first approach leads to a proprietary view of the simulation. The CFD, the CSM solvers are integrated in the workbench and the end user has to learn a specific interface to drive the simulation. The second approach, which is more flexible, is the one we have selected. In this approach, the software bricks, or black-boxes, have a public and standard interface. Then, the end-user would perform the assembly itself and write the control code by himself.
For doing so, the software engineer has to provide

Fig. 1: From Application to software


Application
Simulation
q_{n+1}
$\dot{q}_{n+1}$
$\ddot{q}_{n+1} = \ddot{q}_n + d\ddot{q}_{n+1}$
Software
CSM
CAD
CFD
Control and data tranfers

adapted means to the helicopter scientist. These means can be methods, support, documents, software components, dedicated programming languages or whatever else that would reduce the time the

Presented at the 31st European Rotorcraft Forum, Florence, Italy, September 13-15, 2005.

helicopter scientist will loose during the simulation application writing phase.
In the present work, we have adopted a software strategy that helps to achieve this. The focus point of this strategy is the software component. The component itself is taken into account but also the environment that uses this component. Obviously, these components should offer an interface understandable by the CFD engineer. It should be an interface thought in terms of CFD but not in terms of programming. In the present case, the components developed are based on the CGNS standard [1][2][3][4]. This is a public and standard data representation, used for the input and output description of the data handled by the simulation components.
We describe in the next sections the context of the work, the methodology for building the components and the actual results of the simulation in the field of helicopter blade deformation.

Context

Application Level

The complexity of the helicopter configuration naturally introduces a coupling between the rotor aerodynamics and dynamics. Indeed, compared with fixed wing, helicopter rotor blades have a large number of degrees of freedom due to the articulated hub (or concentrated soft elements), so that, for a given flight condition, the blades motion are unknowns which have to be determined together with the aerodynamic and dynamic characteristics of the rotor. For solving this aeroelastic problem, rotor comprehensive codes consider tbe blade dynamics problem coupled with a simplified aerodynamics model. This kind of methodology allows to self-consistently computing rotor trim, loads, performance and vibrations. The HOST code of Eurocopter, which was used in the present work, is representative of this class of methods [5].
For performing a CFD computation of the helicopter (*elsA*, in our case [6], [7]) and obtain the unsteady aerodynamic field around the rotorcraft, it is necessary to know the blade motion (and eventually the deformation when a soft blade computation is performed) which is prescribed as an input to the CFD analysis. Such a data naturally comes from helicopter comprehensive analysis which trims the rotor towards the desired flight condition, and it can thus be implemented as a time-varying boundary condition in the aerodynamic computation. However, because the aerodynamic description coming from the comprehensive analysis is of much lower level than the CFD one, the distribution of aerodynamic loads and moments along the rotor blades significantly differ, and the detailed aerodynamic field coming out of the CFD analysis is not consistent with the rotor trim as well as with its motion and deformation, as computed in the comprehensive analysis. The only way to render the CFD solution consistent with the blade dynamics and trim is by coupling the CFD code and the rotor comprehensive analysis.

Simulation level

Two types of coupling between CFD and comprehensive analysis can be considered: the weak and the strong coupling. The weak coupling is only valid for steady flight conditions of the helicopter because it assumes 1/rev periodicity for the blade aerodynamics and dynamics. In that case, data can be exchanged between the two codes after each rotor revolution, the CFD computation being performed with prescribed rotor motion and deformation over one blade revolution, the input coming from the comprehensive analysis. The rotor trim is re-computed afterwards by the comprehensive analysis using its internal simplified aerodynamic model corrected from the difference between the CFD and the simplified aerodynamic models for the force and moment distribution over the rotor disk coming from the previous coupling iteration. At convergence, the aerodynamic data used in the rotor trim corresponds to the CFD data. Although a large part of what is presented in this paper also works for this weak coupling approach, it will not be considered anymore in the following.
The strong coupling approach is more general than weak coupling. Indeed, a simultaneous time-marching computation is then performed for both the CFD and the comprehensive analysis, and this last methodology no more uses its internal aerodynamic model. Starting from a trimmed solution with the simplified aerodynamic model of the comprehensive analysis in order to get the blade control angles, the blade dynamic problem is solved by the comprehensive analysis using the aerodynamic forces and moments computed by the CFD. This requires that both the aerodynamic (blades' force and moment distribution) and the dynamic data (blades' motion and deformation) are exchanged between the CFD and the comprehensive analysis at each time step of the computation. A staggered coupling scheme described in [8] provides second-order accuracy for the global fluid-structure coupling process. Up to now, only steady flight conditions of the helicopter were considered. In that case, the time marching process is run until a periodic solution is obtained. However, when this is achieved, it is very likely that the rotor trim conditions are not met because of the simplified aerodynamic model used for trimming.

The trim is obtained by correcting the rotor control angles using a sensitivity analysis obtained with the simplified aerodynamic model of the comprehensive code. A new set of strong coupling time steps has thus to be repeated with these corrected control settings until a new periodic solution is obtained, and the process has to be repeated until a trimmed periodic solution is obtained.

Software level

Former work on code-coupling was completed for helicopters using legacy codes which have been modified for that purpose (see fig.2). The fluid solver *Waves* was used for weak and strong coupling [9] with the *HOST* helicopter dynamics[*]. The coupling was performed using a communication layer developed by the IAG (University of Stuttgart) [10].

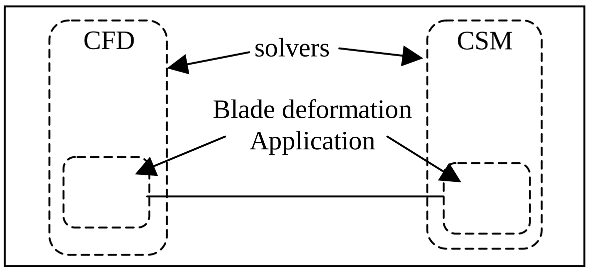


Figure 2: The application as a legacy code modification

This layer provides a proprietary representation of the coupling data as well as a remote communication system based on TCP/IP. The methodology was a direct integration of the libraries and coupling algorithms in the solvers. Thus, only two softwares were involved, the CFD and the CSM. Actually, the development was achieved because the working teams were owner of the source codes and the target application was a hard-coded FORTRAN source.
This is not always the case: more and more code coupling software architecture involves proprietary and commercial solvers. As soon as the customer wants to build an application with at least two codes, a third code appears: the control software (see fig. 3). Most of the time, the controller even has to translate the data coming in and out from both sides of the solvers. This three-code architecture is used for a fluid-structure computation of the helicopter blade deformation. We demonstrate how the use of clearly defined interfaces, using a public data format, has helped us to achieve a better knowledge of the software coupling algorithms and strategies as well as a better maintainability of the resulting application. Indeed, we want to be able to change the structural dynamics code or the fluid solver, without breaking the application itself. Thus, we had to gather the interface information required for our application. We have defined two interfaces, one per solver; the first defines the minimal required set of data the application needs for a complete and consistent interoperability with the structural dynamics solver, the other one is for the CFD solver. Each solver is then seen as a component providing a public interface compliant to the CGNS standard.

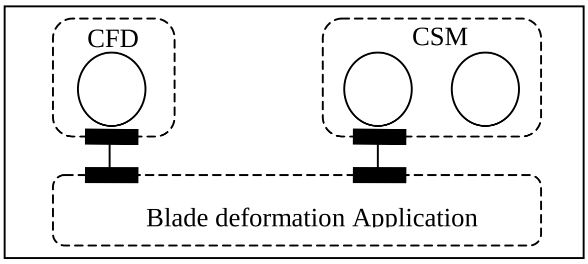


Figure 3: The application as a third component

The third program, the *control*, is in charge of ordering calls to CFD and CSM solvers. This third code also translates data from the CFD format to and from the CSM format. For example, the CFD considers the four blades as a single input and a single output, while the CSM reads and writes data blade per blade. A loop on the four blades is managed by this third program; it splits data for CSM and gathers data for CFD. We also call this third code the "*application*". Actually, we can see this application as the code performing the computation. The two other codes are a means to achieve the goal. We can also call this third code the "*controller*". This controller is in charge of starting, running and synchronizing both solvers.
Such an architecture allows the modification of the coupling algorithm and the translation of data without the knowledge of the solvers internals, but rather using the public interfaces of these.

Data representation

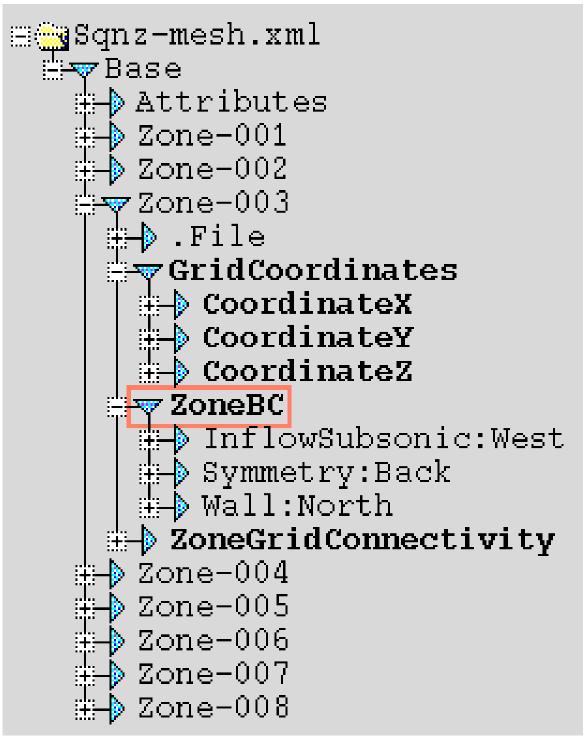
We assume, in a simulation involving a CFD solver, that the CFD data is the main stream of data. We have encapsulated the solvers interfaces in order to have a unified and standard data representation. The standard we used is CGNS, the data representation standard dedicated to the CFD. The fluid data definition is quite simple; it contains a per-blade structure with surface results for forces and moments. It is quite

[*] *The HOST software, initially developed by Eurocopter, provides many functions for simulating the helicopter, including the flight dynamics of the complete rotorcraft. In the present work, HOST is used for computing the rotor structural dynamics and trim, so-called CSM in the remaining part of this paper.*

easy to define such a data structure using the usual CGNS structures.
Now, the data handled by the CSM is not tightly related to CGNS; as a matter of fact, only grid related information can be handled by CGNS. Such a case has been forethought in CGNS and the structure data definition is finally partially taken into account using specific data structures. These are user defined structures, CGNS compliant but actually defined by a proprietary application. For example, we could define and handle results such as `frameTransformMatrix` which describes the rotation from a rigid blade section of the original mesh to the deformed blade section at the azimuth of interest.

CGNS Standard

The computational fluid dynamics community has a data specification standard: CGNS (*CFD General Notation System*) [2][3][4]. This standard has been carried out by NASA in a contract with Boeing, and is now adopted as an AIAA standard [1]. As a matter of fact, it is going to be an ISO international standard within the STEP framework.
More than defining a file format for binary exchange or archival, CGNS truly is a data specification standard. This means its first scope is to define CFD data as a self-contained structure of typed values with a strong semantic. The semantic is the meaning of the data. For example, a data with the name *`Density`* and CGNS compliant carries a heavy implicit meaning for a CFD engineer. The goal of such a self-contained data tree is to specify a CFD context with information involved in simulations, input and output, in order to archive or exchange these simulations. This context includes the mesh information with the coordinates, the connectivities, the boundary conditions, the reference state used for the target configuration and the results, the types of equations used and other computation parameters.
A binary representation on disk of the CGNS tree can be created using the Mid-Level library (MLL), developed and maintained on behalf of the CGNS Steering Committee. It lays now on HDF5 public binary format, which allows CGNS applications to use HDF5 tools or libraries. Other representations of a CGNS tree exist, such as an XML or a Python representation. These representations are individual proposals and are not supported by the Steering Committee. The Python representation is an example of an in-memory CGNS tree, which can be exchanged between interoperable CGNS applications without disk access. Such a use is detailed in the next sections of this paper.

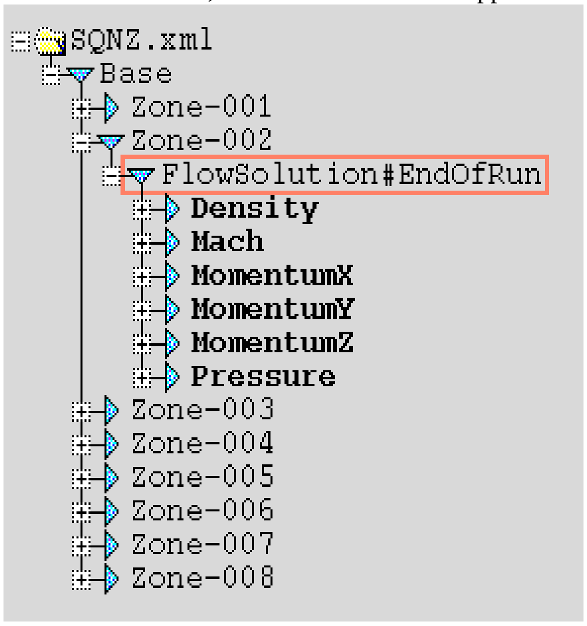

```
Sqnz-mesh.xml
  Base
    Attributes
    Zone-001
    Zone-002
    Zone-003
      .File
      GridCoordinates
        CoordinateX
        CoordinateY
        CoordinateZ
      ZoneBC
        InflowSubsonic:West
        Symmetry:Back
        Wall:North
      ZoneGridConnectivity
    Zone-004
    Zone-005
    Zone-006
    Zone-007
    Zone-008
```
Figure 4: Input data, mesh, BCs and connectivities

The XML allows the use of text based tools for CGNS tree manipulation. The figure 4 and other figures with the same layout are a graphical view of a CGNS data tree. These are views of XML files. These files have been generated from CGNS/HDF5 binary files, but it must be kept in mind that CGNS is a data specification, and then any other data tree representation can be CGNS compliant on a logical point of view

Usual CFD data

The big part of data required as input by a CFD solver usually are the meshes with their connectivities and the fields' initialisation. One can also find smaller data such as the boundary conditions, the physical reference state and the specific data used for the solver setup (e.g. type of equations, multigrid, number of iterations, time integration, etc…). A large part of these data can be defined within the CGNS standard (see fig. 4), which has been designed in order to cover a large range of CFD applications.
The grids can be defined as structured or unstructured grids, and related definitions are made in a way that allows a large common part in the *structured* and *unstructured* types of them. It is then quite easy to use structured meshes as an unstructured mesh with hexaedric elements for example. The connectivity and the boundary condition patches can be defined as a list of points, which are valid in either structured or unstructured types of grids.

The output data one can find in a CGNS file are field data (see fig. 5), such as *`MomentumX`* or *`Density`*, but also discrete data or solver specific data (such as convergence history). The standard allows a user defined structure, which can be used for all kind of data, input as well as output. A user specific data is CGNS compliant, but obviously it can be understood only by the applications that know about the meaning of the structure. A common example, as shown in fig. 6, is the solver specific parameters. One can find parameters that already exist in CGNS, *`gamma`* and *`SpecificHeatRatio`* for example. While not recommended, such a duplication of information has been used by our solver for migration purpose. As soon as a solver or an application is ready to read the CGNS information, it does. Otherwise the application



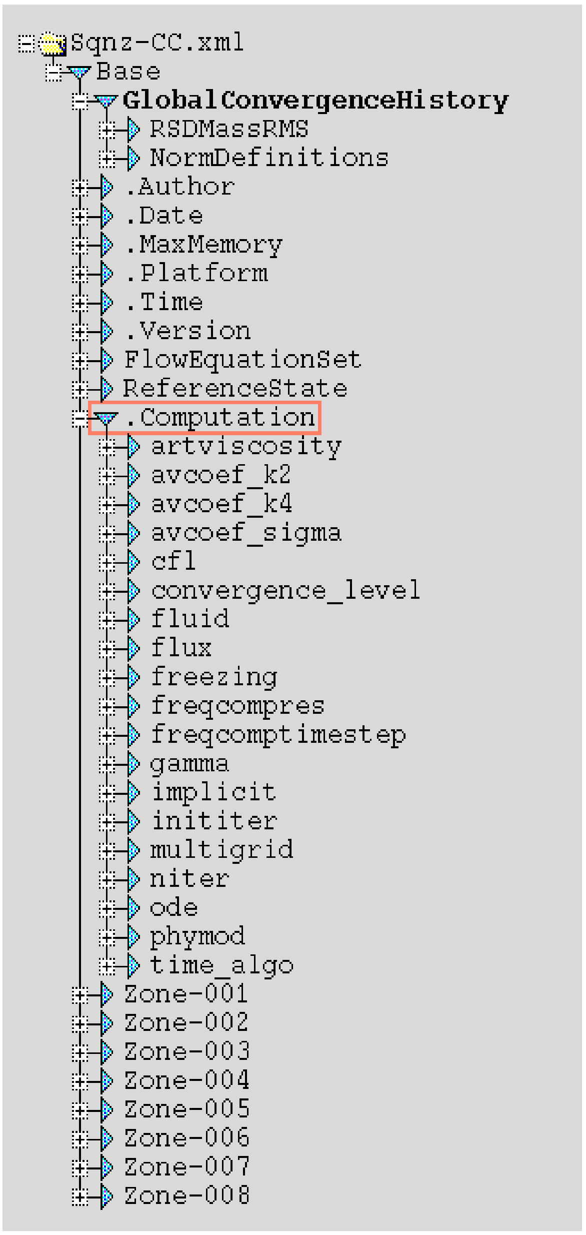
```
SQNZ.xml
  Base
    Zone-001
    Zone-002
      FlowSolution#EndOfRun
        Density
        Mach
        MomentumX
        MomentumY
        MomentumZ
        Pressure
    Zone-003
    Zone-004
    Zone-005
    Zone-006
    Zone-007
    Zone-008
```

Figure 5: Flow Solution Fields

The output identified as `FlowSolution#EndOfRun` contains fields with reserved names (`Density`, `Mach`…) but the name of the solution itself is defined by the application. The `FlowSolution#EndOfRun` name is a private agreement set between our proprietary CFD applications. This private rule added to the CGNS standard belongs to our CGNS profile. Another rule you can guess on the figures is "*All user defined node names should start with a dot*".

CGNS compliant proprietary extensions

The CGNS standard is an agreement between CFD community people over the world. This includes end users coming from very large aerospace companies, research centres, universities, software companies…



```
Sqnz-CC.xml
  Base
    GlobalConvergenceHistory
      RSDMassRMS
      NormDefinitions
    .Author
    .Date
    .MaxMemory
    .Platform
    .Time
    .Version
    FlowEquationSet
    ReferenceState
    .Computation
      artviscosity
      avcoef_k2
      avcoef_k4
      avcoef_sigma
      cfl
      convergence_level
      fluid
      flux
      freezing
      freqcompres
      freqcomptimestep
      gamma
      implicit
      inititer
      multigrid
      niter
      ode
      phymod
      time_algo
    Zone-001
    Zone-002
    Zone-003
    Zone-004
    Zone-005
    Zone-006
    Zone-007
    Zone-008
```

Figure 6: Solver Specific Parameters

Thus, CGNS is the largest consensus we have found amongst these organisations. It cannot describe all possible data a CFD application can use or produce, it describes the common set on which there is an agreement. It is a basis for applications that want to share data with other applications. For us, at ONERA, it is more than that: CGNS is the basis for CFD data specification, including code-coupling data [11]. Even if a CFD solver has its own internal representation (and they obviously have such a representation), this solver has to provide a CGNS view of its public data. This allows a better interoperability but also a better understanding of the solver IO. Any kind of information going from a CFD tool to another application should be defined with CGNS. The definition has to be understood as a

logical definition, with a meaning, more than a file binary format.
As much information cannot be defined with dedicated CGNS types in the standard, at this stage we have at least two strategies:

1. Use the free text nodes, store specific data in a private text format.
2. Create a sub-tree with user defined nodes and typed data.

We think the first one is good in solving data interoperability for a short term application; the second one is the best in the long term range. With the definition a typed sub-tree, we can publish a public view of our data. Even if this sub-tree is dedicated to our set of applications, it lays on a documented and publicly available data specification. Moreover, we can submit our extensions to the steering committee and see these become the standard.
In fig. 6 we show the `.Computation` sub-tree, it contains all the fluid solver dedicated parameters. The nodes can contain textual or numerical values, their meaning is only known by the application. At the physical level, i.e. using such a tree on a disk, any CGNS application can read them, get the values using CGNS tools and libraries.

Grammar restriction

The CGNS data specification is driven by a grammar, this is the SIDS grammar. It defines a set of possible data trees, with all allowed options and extensions. However, a given application would not use all the possibilities of the grammar. Obviously a structured mesh solver would not take into account the

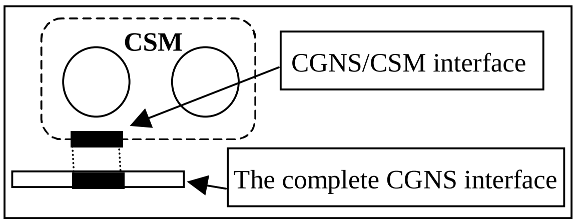


Figure 7: CGNS input restriction

unstructured meshes or any kind of information related to unstructured meshes. Thus, the actual use of CGNS in this structured solver is restricted to the structured restriction of the grammar.
Rather than a blind methodology, trying to map data coming from a tool to another one, our CGNS component approach focuses on the real meaning of the required data. This leads to both a better understanding of the component and its related services, but also in a better understanding of the whole simulation involving all components. For example, if a data cannot be set as a CGNS data, we ask ourselves if we actually require this data for our simulation. Sometimes the data can be deduced from other data, or it appears this was not the relevant data.
The CSM input data should be understood by the CSM solver (see fig. 7). As our data are CGNS compliant, we have to find out the required subset of the standard. Such a definition has to be done for the output data too (see fig. 8). These two restrictions can be melt in a single one, it is the component restriction to the CGNS interface. It guarantees that any CSM component input or output is CGNS compliant [12].
There are cases where the data is not defined in CGNS. For example the blade section force. In that case, the user defined structure is used, but the CSM component still is CGNS compliant. As the data we are dealing with are 90% CFD data, it is better to use

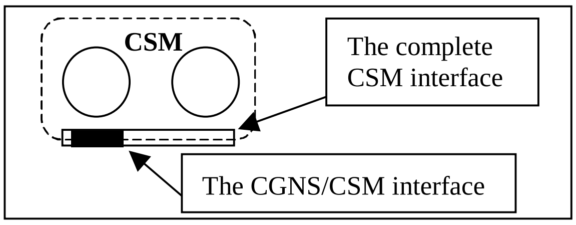
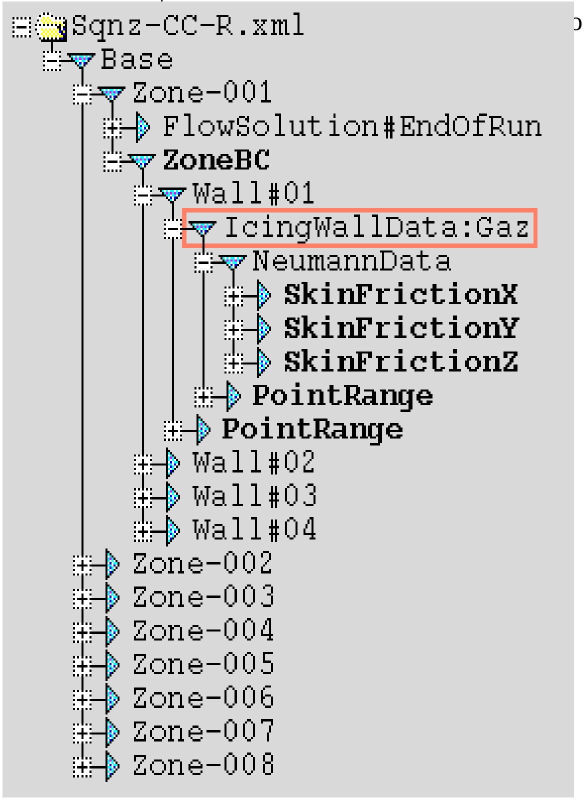


Figure 8: CGNS output restriction

a CFD standard and adapt it to specific non-CFD data than to use a completely anonymous data format. In that latter case, we would have to redefine 90% of our



```
Sqnz-CC-R.xml
  Base
    Zone-001
      FlowSolution#EndOfRun
      ZoneBC
        Wall#01
          IcingWallData:Gaz
            NeumannData
              SkinFrictionX
              SkinFrictionY
              SkinFrictionZ
            PointRange
          PointRange
        Wall#02
        Wall#03
        Wall#04
    Zone-002
    Zone-003
    Zone-004
    Zone-005
    Zone-006
    Zone-007
    Zone-008
```

Figure 9: Boundary Condition Output

Once we have the CGNS grammar restrictions for a given component, we can connect components together. Some connections require a translation or data reorganization. This is performed by the application code so-called *controller*.

Example tree

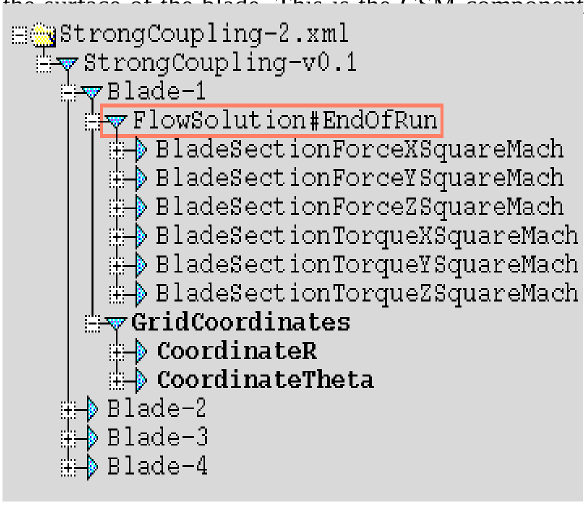
We show in fig. 9 an example of a code-coupling data exchange. In that case which deals with icing, the skin friction is the data of interest. The tree contains a set of boundary conditions used to host the coupling-data, in input as well as output. Each boundary condition contains an application patch (`PointRange`) and an array of values (`SkinFriction`). These BCs are the data interfaces between codes providing the data and codes using it.

Component architecture

Our experiences in code coupling have shown that some parts of the whole software system are quite stable while other parts have to be changed for every application target. A factorization of these codes leads to a set of solvers and remaining lines of codes that could not be generalized for any kind of application. These remaining parts are so-called the application or third code. The requirements of this third code are (1) the ability to quickly perform an assembly of solvers, (2) an ability to translate and operate on data and finally (3) a powerful way to control the functions calls. The python programming language suits perfectly with these definitions. Moreover, the language provides a packaging methodology that makes it easier to deliver, to deploy and to integrate components. Our components are python packages.
The controller is a Python software. It uses both fluid and structure solvers as imported modules. Actually, the structure solver is a remote one but this feature is transparent to the user. The fluid and the structure transient data trees are stored in memory, there is no file exchange. The memory representation of the CGNS tree is based on the Python data structure, that is a tree of nodes where a node is a list composed of a name (string), a value (Numeric array), a list of children nodes (list) and the node type (string). Specific software drivers are translating the in-memory CGNS tree to the solver dedicated data structure (i.e. C++ for CFD solver and C buffer for Structure solver).

Fluid component

The fluid solver is the *elsA* software. It provides a Python/CGNS interface and no modifications have been necessary for the component integration. The interface required for our application was mostly limited to the blade force and torque spanwise distribution. At each computation step of the strong coupling process the fluid solver sends a data tree containing the data for the four blades (see fig. 10). The expected input is the motion of every points of the surface of the blade. This is the CSM component



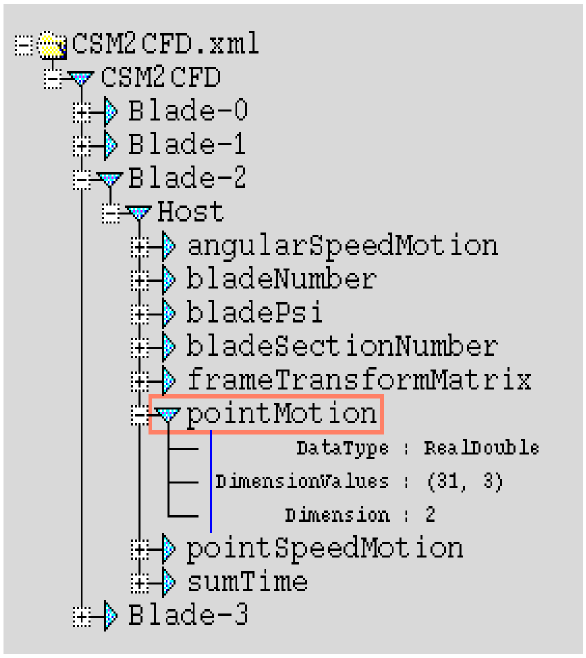
```
StrongCoupling-2.xml
  StrongCoupling-v0.1
    Blade-1
      FlowSolution#EndOfRun
        BladeSectionForceXSquareMach
        BladeSectionForceYSquareMach
        BladeSectionForceZSquareMach
        BladeSectionTorqueXSquareMach
        BladeSectionTorqueYSquareMach
        BladeSectionTorqueZSquareMach
      GridCoordinates
        CoordinateR
        CoordinateTheta
    Blade-2
    Blade-3
    Blade-4
```

Figure 10: Fluid Component Output

Our CFD solver is able to modify the grids, but we can also build an application using another external component. Our open architecture allows the addition of a mesh deformation component inserted between the CSM and the CFD. Then the CFD solver would receive as well a new mesh as input instead of computing it from the exchanged data.

CSM component

We had to make an integration effort for the CSM component. This one was not available as a python package and it has no CGNS compliant interface. An encapsulation of the CSM code was done. Actually, the encapsulation was performed on the client part of the IAG interface. The IAG interface allows a transparent remote communication.



```
CSM2CFD.xml
  CSM2CFD
    Blade-0
    Blade-1
    Blade-2
      Host
        angularSpeedMotion
        bladeNumber
        bladePsi
        bladeSectionNumber
        frameTransformMatrix
        pointMotion
                 DataType : RealDouble
          DimensionValues : (31, 3)
                Dimension : 2
        pointSpeedMotion
        sumTime
    Blade-3
```

Figure 11: Structural Component Output

Control component

The third code can also be seen as a component, while it is really dedicated to an application and hardly reusable for another simulation.
The control is a python script, it starts the solver processes, it checks files and other restart conditions and finally enters into a loop. This loop calls the solvers sequencially. Each solver waits until the control asks it to perform the next iteration.
When the CFD solver sends the data to the control, this latter splits the four blades data in four parts and asks the CSM to perform the computation one blade per one and stores the output data. Once the four computations are done the control merges the four output data in a single tree and sends it to the CFD solver. The CSM has the responsibility to finish the simulation, in other words it finishes its computation and the control asks CFD to stop.

Application

During the simulation, the Python interpreter imports the control script at each iteration exchange. It is possible to change this script in order to change the behaviour of this controller during the computation. The simplest change is the request to write the in-memory data. Such data storage can help for debug purpose or in order to archive a set of input/output data for a simulation stub replay.
An example of strong coupling is presented here for the case of the 7A rotor at high-speed forward flight (μ=0.4, 200C_T/σ=12.5), which corresponds to a test condition in the ONERA S1 Modane wind-tunnel. More examples of applications are shown in [13]. The present computation is inviscid, and no re-trim was completed at the end of the strong coupling. In this computation, the first revolutions were completed by taking into account the three components of the blade sectional force and the pitching moment component of the sectional moments only, the other components being set to zero. The computation was re-run afterwards with all three components of moment taken into account in the computations. The total number of blade revolutions computed is then equal to 12.
The convergence of the sectional lift close to the blade tip for the 6 last blade revolutions is plotted in figure 12, and the corresponding convergence for the pitching moment in figure 13. As can be seen, the convergence is satisfactory for both quantities. A comparison of the blade to blade lift evolution for the last rotor revolution (see fig. 14) also confirms that good periodicity conditions have been obtained.

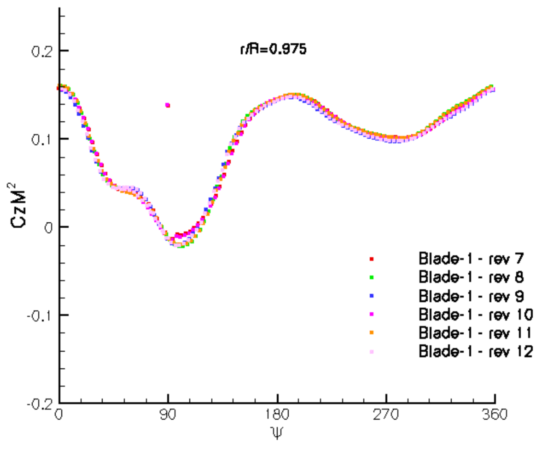


Figure 12: Convergence of lift at the blade tip for the last 6 revolutions

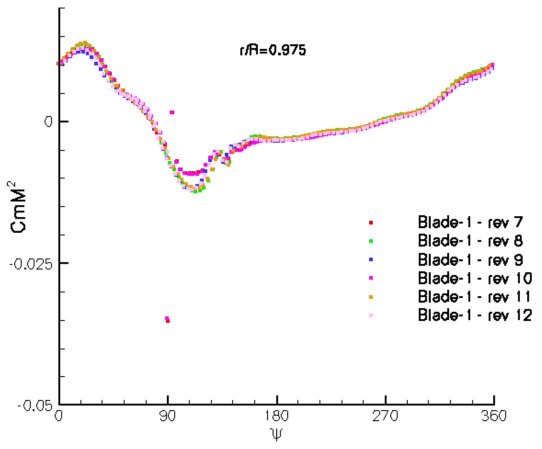


Figure 13: Convergence of pitching moment at the blade tip for the last 6 revolutions

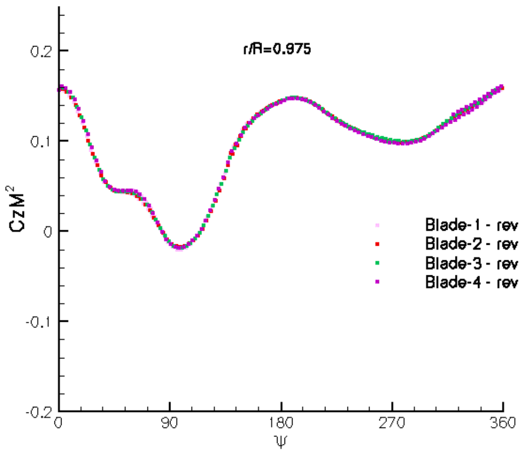


Figure 14: Convergence of lift at the blade tip for the 4 blades during the last revolution

The evolution of the non-dimensional rotor lift is plotted in figure 15 for a HOST alone computation, with its internal simplified aerodynamic model, a strong coupling computation taking into account the three force components and the pitching moment only, and a strong coupling computation taking into account the six components of the aerodynamic force and moment. As expected, these last two quantities hardly differ, because the main component in the aerodynamic moment is the pitching moment. All three computations show a 4/rev periodicity of the lift, which is due to the 4-bladed rotor. The mean value of thrust is slightly smaller for the HOST-*elsA* coupling, because the coupled solution was not re-trimmed to the required value. This last point is also clear when looking at the trim condition which is imposed in S1 Modane: $\beta_{1c} + \theta_{1s} = 0$. While this condition is clearly met by the HOST alone computation (see fig. 16), this is obviously no more the case for the coupled computation, so that a new trim would be necessary together with a new strong coupling computation in order to correctly simulate the test case. A phase difference is also found between the HOST alone and the coupled computation of lift, which is probably introduced by the unsteady three-dimensional aerodynamics.

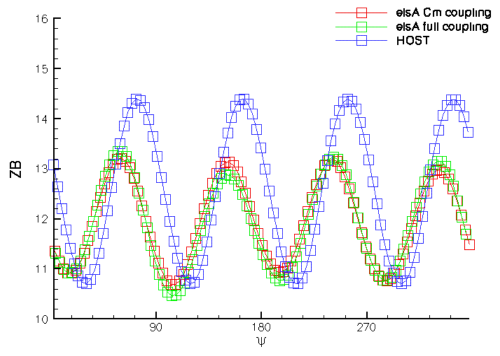


Figure 15: Comparison of rotor thrust computed by HOST and by HOST-elsA

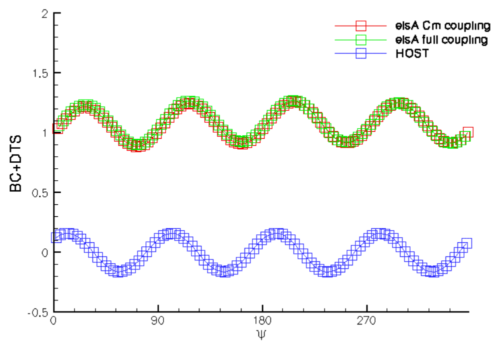


Figure 16: Comparison of trim condition computed by HOST and by HOST-elsA

Conclusion

The helicopter blade fluid-structure interaction application is a complex simulation involving several computer codes. The HOST software, developed by Eurocopter, gathers the accumulated know how of the company for computing the global helicopter characteristics, and more particularly the blade fluid-structure coupling. Indeed, HOST includes solvers for aerodynamics, flight mechanics, structural dynamics, and other helicopter dedicated simulation functions. In our application, we have changed the HOST fluid solver used for the blades by our own external CFD solver. The strong coupling between the two separate softwares was achieved using a new development methodology involving the CGNS standard for public interface definition.

Such an approach has proved its efficiency and can actually be used for any kind of CFD related codes. In the future, existing CFD tools will be migrated and new developments will follow this CGNS-component strategy.

Acknowledgments

The authors are willing to acknowledge the French Official Services (DPAC, DGA) for funding the CHANCE project.

References

1. CGNS Team, *“Standard Interface Data Structures”*, AIAA-R-101A-2005
2. Poirier D.M.A., Allmaras D.R., McCarthy D.R., Smith M.F., Enomoto F.Y., *"The CGNS System"* , AIAA Paper 98-3007
3. Poirier D.M.A., Bush R.H., Cosner R.R., Rumsey C.L., McCarthy D.R. ,*"Advances in the CGNS Database Standard for Aerodynamics and CFD"*, AIAA Paper 2000-0681
4. Legensky S.M., Edwards D.E., Bush R.H, Poirier D.M.A., Rumsey C.L., Cosner R.R., Towne C.E., *"CFD General Notation System (CGNS): Status and Future Directions"*, AIAA Paper 2002-0752
5. Benoit, B., Dequin, A-M., Kampa, K., Grunhagen, W., Basset, P-M., Gimonet, B.: “HOST: A General Helicopter Simulation Tool for Germany and France”, American Helicopter Society, 56th Annual Forum, Virginia Beach, Virginia, May 2000.
6. Cambier, L., Gazaix, M., “elsA : An Efficient Object-Oriented Solution to CFD Complexity”, 40th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada (January 14-17, 2002)
7. Boniface, J.-C., Cantaloube, B., Jollès, A., “Rotorcraft simulations using an object oriented approach”, 26th European Rotorcraft Forum, The Hague, The Netherlands (September 26-29, 2000)
8. Altmikus A., Wagner S., *On the timewise accuracy of staggered aeroelastic simulations of rotary wings,*AHS Aerodynamics, Acoustics and Test and Evaluation Technical Specialists Meeting, San Francisco, January 23-25, 2002
9. Servera G., Beaumier P.,Costes M., *“A weak coupling method between the dynamics code HOST and the 3D unsteady Euler code WAVES”*, 26th European Rotorcraft Forum, Den Hague, 26-29 Sept. 2000
10. Altmikus A., Wagner S., Hablowetz T., Well K.,*”On the accuracy of modular aeroelastic methods applied to fixed and rotary wings”*, 18th AIAA Applied Aerodynamics Conference, Denver, Aug. 14-17, 2000
11. Poinot M., Rumsey C.L., Mani M., *"Impact of CGNS on CFD workflow”*, AIAA-2004-2142
12. Poinot M., Montreuil E., Hénaux E., *"Checking CFD interfaces in a multi-disciplinary workflow with an XML/CGNS compiler”*, AIAA-2005-1155
13. Beaumier P., Costes M., Rodriguez B., Poinot M., Cantaloube B., ‘*Weak and strong coupling between the elsA CFD solver and the HOST helicopter comprehensive analysis’*, 31st ERF, Florence 2005.